Version
6
5.1
5.0
4.0
3.0

About ForEach

11/27/2017 4 minutes to read Contributors all

In this article

SHORT DESCRIPTION

Describes a language command you can use to traverse all the items in a collection of items.

LONG DESCRIPTION

The Foreach statement (also known as a Foreach loop) is a language construct for stepping through (iterating) a series of values in a collection of items.

The simplest and most typical type of collection to traverse is an array. Within a Foreach loop, it is common to run one or more commands against each item in an array.

Syntax

The following shows the ForEach syntax:

syntax Copy

```
foreach ($<item> in $<collection>){<statement list>}
```

The part of the `ForEach` statement enclosed in parenthesis represents a variable and a collection to iterate. PowerShell creates the variable `$<item>` automatically when the `Foreach` loop runs. Prior to each iteration through the loop, the variable is set to a value in the collection. The block following a `Foreach` statement `{<statement list>}` contains a set of commands to execute against each item in a collection.

Examples

For example, the `Foreach` loop in the following example displays the values in the `$letterArray` array.

PowerShell Copy

```
$letterArray = "a","b","c","d"
foreach ($letter in $letterArray)
{
  Write-Host $letter
}
```

In this example, the `$letterArray` array is created and initialized with the string values `"a"`, `"b"`, `"c"`, and `"d"`. The first time the `Foreach` statement runs, it sets the `$letter` variable equal to the first item in `$letterArray` (`"a"`). Then, it uses the `Write-Host` cmdlet to display the letter a. The next time through the loop, `$letter` is set to `"b"`, and so on. After the `Foreach` loop displays the letter d, PowerShell exits the loop.

The entire `Foreach` statement must appear on a single line to run it as a command at the PowerShell command prompt. The entire `Foreach` statement does not have to appear on a single line if you place the command in a .ps1 script file instead.

`Foreach` statements can also be used together with cmdlets that return a collection of items. In the following example, the Foreach statement steps through the list of items that is returned by the `Get-ChildItem` cmdlet.

PowerShell Copy

```
foreach ($file in Get-ChildItem)
{
  Write-Host $file
}
```

You can refine the example by using an `If` statement to limit the results that are returned. In the following example, the `Foreach` statement performs the same looping operation as the previous example, but it adds an `If` statement to limit the results to files that are greater than 100 kilobytes (KB):

PowerShell Copy

```
foreach ($file in Get-ChildItem)
{
```

```
  if ($file.length -gt 100KB)
  {
    Write-Host $file
  }
}
```

In this example, the `Foreach` loop uses a property of the `$file` variable to perform a comparison operation (`$file.length -gt 100KB`). The `$file` variable contains all the properties in the object that is returned by the `Get-ChildItem` cmdlet. Therefore, you can return more than just a file name. In the next example, PowerShell returns the length and the last access time inside the statement list:

PowerShell Copy

```
foreach ($file in Get-ChildItem)
{
  if ($file.length -gt 100KB)
  {
    Write-Host $file
    Write-Host $file.length
    Write-Host $file.lastaccesstime
  }
}
```

In this example, you are not limited to running a single command in a statement list.

You can also use a variable outside a `Foreach` loop and increment the variable inside the loop. The following example counts files over 100 KB in size:

PowerShell Copy

```
$i = 0
foreach ($file in Get-ChildItem) {
  if ($file.length -gt 100KB) {
    Write-Host $file "file size:" ($file.length / 1024).ToString("F0") KB
    $i = $i + 1
  }
}

if ($i -ne 0) {
  Write-Host
  Write-Host $i " file(s) over 100 KB in the current directory."
}
else {
  Write-Host "No files greater than 100 KB in the current directory."
}
```

In the preceding example, the `$i` variable is set to `0` outside the loop, and the variable is incremented

inside the loop for each file that is found that is larger than 100 KB. When the loop exits, an `If` statement evaluates the value of `$i` to display a count of all the files over 100 KB. Or, it displays a message stating that no files over 100 KB were found.

The previous example also demonstrates how to format the file length results:

PowerShell Copy

```
($file.length / 1024).ToString("F0")
```

The value is divided by 1,024 to show the results in kilobytes rather than bytes, and the resulting value is then formatted using the fixed-point format specifier to remove any decimal values from the result. The 0 makes the format specifier show no decimal places.

In the following example, the function defined parses PowerShell scripts and script modules and returns the location of functions contained within. The example demonstrates how to use the `MoveNext` method (which works similarly to `skip X` on a `For` loop) and the `Current` property of the `$foreach` variable inside of a foreach script block. The example function can find functions in a script even if there are unusually- or inconsistently-spaced function definitions that span multiple lines.

PowerShell Copy

```
function Get-FunctionPosition {
  [CmdletBinding()]
  [OutputType('FunctionPosition')]
  param(
    [Parameter(Position = 0, Mandatory,
      ValueFromPipeline, ValueFromPipelineByPropertyName)]
    [ValidateNotNullOrEmpty()]
    [Alias('PSPath')]
    [System.String[]]
    $Path
  )

  process {
    try {
      $filesToProcess = if ($_ -is [System.IO.FileSystemInfo]) {
        $_
      } else {
        $filesToProcess = Get-Item -Path $Path
      }
      $parser = [System.Management.Automation.Language.Parser]
      foreach ($item in $filesToProcess) {
        if ($item.PSIsContainer -or

            $item.Extension -notin @('.ps1', '.psm1')) {
          continue
        }
```

```
        $tokens = $errors = $null
        $ast = $parser::ParseFile($item.FullName, ([REF]$tokens),
          ([REF]$errors))
        if ($errors) {
          $msg = "File '{0}' has {1} parser errors." -f $item.FullName,
            $errors.Count
          Write-Warning $msg
        }
        :tokenLoop foreach ($token in $tokens) {
          if ($token.Kind -ne 'Function') {
            continue
          }
          $position = $token.Extent.StartLineNumber
          do {
            if (-not $foreach.MoveNext()) {
              break tokenLoop
            }
            $token = $foreach.Current
          } until ($token.Kind -in @('Generic', 'Identifier'))
          $functionPosition = [pscustomobject]@{
            Name       = $token.Text
            LineNumber = $position
            Path       = $item.FullName
          }
          Add-Member -InputObject $functionPosition `
            -TypeName FunctionPosition -PassThru
        }
      }
    }
    catch {
      throw
    }
  }
}
```

SEE ALSO

about_Automatic_Variables

about_If

ForEach-Object